# A MECHANISM FOR MAPPING JAVA OBJECTS
# ONTO AN LDAP REPOSITORY

**Field of the Invention**

The invention relates to data storage, and more specifically to standardized storage of object attributes.

**Background of the Invention**

Object-oriented programming is a method of programming that pairs programming tasks and data into re-usable chunks known as objects. Each object comprises attributes (*i.e.*, data) that define and describe the object. Java classes are meta-definitions that define the structure and characteristics for groups of objects in object-oriented parlance. Java classes when instantiated create instances of the Java classes and are then considered Java objects. Methods within Java objects are used to get or set attributes of the Java object and to change the state of the Java object. Associated with each method is code that is executed when the method is invoked.

Java objects of an application, when instantiated, are resident within a computer's memory (*e.g.*, ROM). However, when the computer is turned off or restarted, the objects are flushed from the memory. As such, the objects need to be loaded back into the computer's memory in order to be accessed. Consequently, there is a need for the objects and, particularly, the object attributes to be stored in persistent repository.

The Java objects may be stored as basic system files with a flat file structure. Basic systems files are a simple way of storing the Java objects that would be easily accessible to the application from which the Java objects originated. Unfortunately, other applications would need to learn the application's application programming interfaces ("APIs") in order to be able to access the object data from the basic system files. Having to learn these APIs is generally a deterrent that would effectively prevent the basic systems files from being usable outside of the originating application.

Storing the Java objects according to a standard protocol would seem to be a necessary improvement. Indeed, there are known methods for serializing Java objects and their attributes and storing them in a Lightweight Directory Access Protocol ("LDAP") repository. LDAP is an industry standard for storing information into a quickly accessible database. Serializing a Java object comprises

1     converting the Java object's physical memory representation into a string of

2     characters and storing the string of characters in the LDAP repository.

3         Unfortunately, simply serializing the Java objects and storing them in a

4     LDAP repository does not make the stored Java objects readable or useable by other

5     applications that do not have the same Java object structures. The string of

6     characters has to be converted into the Java object from which it originated in order

7     to be readable. Moreover, serializing the whole Java object necessarily stores

8     attributes that are unimportant to other applications (*e.g.*, attributes that represent

9     internal operations of the Java object).

10        LDAP does include a plurality of data objects including different types of

11     attributes (*e.g.*, commonName, description, ipHostNumber, host, uidNumber).

12     These LDAP data objects and their attributes, unfortunately, may not directly

13     correspond to an application's Java objects and their attributes. In order to be

14     usable, the LDAP data objects need to be mapped to Java objects. New LDAP data

15     objects and attributes need to be created. If the Java objects are mapped to LDAP

16     data objects, then the Java objects may be stored in a LDAP repository in a manner

17     that is readable and useable by other applications.

18 **Summary of the Invention**

19        A method and apparatus for dynamically storing Java objects in a LDAP

20     repository in a manner useable to other applications are disclosed. According to an

21     embodiment of the invention, a persistent data manager maps certain persistent

22     attributes of Java objects to corresponding LDAP attributes. The certain persistent

23     attributes are preferably chosen as attributes that are of interest to other outside

24     applications. The LDAP attributes are named with a syntax that easily identifies the

25     attributes. The persistent data manager is preferably a software component that

26     executes processes to dynamically determine the persistent attributes of the Java

27     objects as well as the path or distinguished name ("dn") of the corresponding LDAP

28     attributes. The persistent data manager also preferably uses reflection, a known

29     Java technique, to determine the persistent attribute values. The persistent data

30     manager preferably invokes the LDAP API(s) necessary, either directly or indirectly

31     through a utility class, to read and write the persistent attribute values from and to

32     the LDAP repository.

33        These and other advantages are achieved by a method for mapping objects

34     onto a lightweight directory access protocol repository, comprising requesting that

1  an object be stored in a lightweight directory access protocol ("LDAP") repository,

2  wherein the object includes attributes and is used in an object-oriented

3  programming application, retrieving a list of persistent attributes from the object,

4  wherein the persistent attributes are a subset of the attributes and the persistent

5  attributes each comprise a persistent attribute value, determining a path, wherein the

6  path identifies a location in the LDAP repository, retrieving the persistent attribute

7  values from the object, and storing the object in the LDAP repository so that the

8  persistent attributes are stored in a format that is useable to applications other than

9  the object-oriented programming application.

10  Likewise, these and other advantages are achieved by a method for

11  retrieving objects mapped onto a lightweight directory access protocol repository,

12  comprising, requesting that an object be retrieved from a lightweight directory

13  access protocol ("LDAP") repository, wherein the object includes attributes and is

14  used in an object-oriented programming application, retrieving a list of persistent

15  attributes from the object, wherein the persistent attributes are a subset of the

16  attributes and the persistent attributes each comprise a persistent attribute value,

17  determining a path, wherein the path identifies a location in the LDAP repository,

18  retrieving the persistent attribute values from the location in the LDAP repository

19  identified by the path, and setting the persistent attributes in the object with the

20  retrieved persistent attribute values.

21  These and other advantages are also achieved by a computer readable

22  medium containing instructions for mapping objects onto a lightweight directory

23  access protocol repository, by requesting that an object be stored in a lightweight

24  directory access protocol ("LDAP") repository, wherein the object includes

25  attributes and is used in an object-oriented programming application, retrieving a

26  list of persistent attributes from the object, wherein the persistent attributes are a

27  subset of the attributes and the persistent attributes each comprise a persistent

28  attribute value, determining a path, wherein the path identifies a location in the

29  LDAP repository, retrieving the persistent attribute values from the object, and

30  storing the object in the LDAP repository so that the persistent attributes are stored

31  in a format that is useable to applications other than the object-oriented

32  programming application.

33  A computer system that supports mapping objects onto a lightweight

34  directory access protocol repository, comprising a lightweight directory access

1     protocol ("LDAP") repository, a processor that runs an object-orient programming

2     application, wherein the object-oriented programming application generates, an

3     object, wherein the object includes attributes and is used in an object-oriented

4     programming application, a persistent data manager, that acts as a layer between the

5     object and the LDAP repository, wherein the persistent data manager stores the

6     object in the LDAP repository by, retrieving a list of persistent attributes from the

7     object, wherein the persistent attributes are a subset of the attributes and the

8     persistent attributes each comprise a persistent attribute value, determining a path,

9     wherein the path identifies a location in the LDAP repository, retrieving the

10     persistent attribute values from the object, and storing the object in the LDAP

11     repository so that the persistent attributes are stored in a format that is useable to

12     applications other than the object-oriented programming application.

**Brief Description of the Figures**

14         Figure 1 is a block diagram of a network system in which an embodiment of

15     the present invention is used.

16         Figure 2 is a block diagram conceptually illustrating a persistent data

17     manager, and the persistent data manager's operation, according to an embodiment

18     of the present invention

19         Figure 3 is a directory tree illustrating an exemplary directory structure of a

20     LDAP repository mapped to by a persistent data manager according to an

21     embodiment of the present invention.

22         Figures 4a and 4b are a static structure diagram illustrating an

23     implementation class of a persistent data manager, and operation of the persistent

24     data manager, according to an embodiment of the present invention.

25         Figures 5a and 5b are flowcharts illustrating methods for storing and reading

26     Java objects in and from a LDAP data repository according to an embodiment of the

27     present invention.

28         Figures 6a and 6b are sequence diagrams illustrating exemplary processes

29     for storing and reading a node group object in and from a LDAP data repository

30     according to an embodiment of the present invention.

**Detailed Description of the Invention**

32         The present invention may be used with computer systems that utilize

33     object-oriented applications and need to store data (*e.g.*, object attributes) in a

34     format that is readable by different applications. Figure 1 illustrates a computer

1  network system 10 with which the present invention may be used. The network

2  system 10 comprises a ServiceControl Manager ("SCM") 12 running on a Central

3  Management Server ("CMS") 14 and one or more nodes 16 managed by the SCM

4  12 on the CMS 14. Together the one or more nodes 16 managed by the SCM 12

5  make up an SCM cluster 17. A group of nodes 16 may be organized as a node

6  group 18.

7  The CMS 14 preferably is an HP-UX 11.x server running the SCM 12

8  software. The CMS 14 includes a memory (not shown), a secondary storage device

9  141, a processor 142, an input device (not shown), a display device (not shown),

10  and an output device (not shown). The memory, a computer readable medium, may

11  include, RAM or similar types of memory, and it may store one or more

12  applications for execution by processor, including the SCM 12 software. The

13  secondary storage device 141, a computer readable medium, may include a hard

14  disk drive, floppy disk drive, CD-ROM drive, or other types of non-volatile data

15  storage.

16  The processor 142 executes the SCM 12 software and other application(s),

17  which are stored in memory 143 or secondary storage 141, or received from the

18  Internet or other network 24, in order to provide the functions and perform the

19  methods described in this specification, and the processing may be implemented in

20  software, such as software modules, for execution by the CMS 14 and modes 16. In

21  addition to the that described above, the CMS 14 preferably also comprises a data

22  repository 26 for the SCM cluster 17, a web server 28 that allows web access to the

23  SCM 12 and a depot 30 comprising products used in the configuring of nodes, and a

24  I/UX server 32. See ServiceControl Manager Technical Reference, HP part

25  number: B8339-90019, which is hereby incorporated by reference and which is

26  accessible at http://www.software.hp.com/products/scmgr, for a more detailed

27  description of the CMS 10 and the SCM 12.

28  Referring to Figure 1, the SCM 12 preferably supports managing a single

29  SCM cluster 17 from a single CMS 14. All tasks performed on the SCM cluster 17

30  are initiated on the CMS 14 either directly or remotely, for example, by reaching the

31  CMS 14 via a web connection 20. Therefore, a workstation 22 at which a user sits

32  may use a web connection 20 over a network 24 to the CMS 14 in order to perform

33  tasks on the SCM cluster 17. The workstation 22 preferably comprises a display, a

34  memory, a processor, a secondary storage, an input device and an output device.

The nodes 16 are preferably HP-UX servers or other servers and they may be referred to as "*managed nodes*" or simply as "*nodes*". The concept of a node 16 is that it represents a single instance of HP-UX running on some hardware. The node 16 may comprise a memory, a secondary storage device, a processor, an input device, a display device, and an output device.

Although the CMS 14 is depicted with various components, one skilled in the art will appreciate that this server can contain additional or different components. In addition, although aspects of an implementation consistent with the present invention are described as being stored in memory, one skilled in the art will appreciate that these aspects can also be stored on or read from other types of computer program products or computer-readable media, such as secondary storage devices, including hard disks, floppy disks, or CD-ROMs; a carrier wave from the Internet or other network; or other forms of RAM or ROM. The computer-readable media may include instructions for controlling the CMS 14 (and/or the nodes 16) to perform a particular method, such as those described herein.

The SCM 12 is an application preferably programmed in Java that operates in a Java object-oriented environment. The SCM 12 comprises objects (*e.g.*, Java objects) that provide the functionality of the SCM 12. In the system 10, each user, node, node group, role, tool, authorization, user name, node name, and node group name is, for each instance, represented by a single Java object. A role defines the role (*e.g.*, administrator, database manager, web manager, etc.) a user may have on a certain node(s) or node group(s), where each role has one or more tools associated with it that a user with the role may execute. A tool is an executable(s) that performs a process. An authorization defines the node(s) and node group(s) a user is authorized to access and what roles the user has on the authorized node(s) or node group(s).

When the attributes of any of the above (*i.e.*, user, node, node group, etc.) are changed or need to be accessed, the representative object is instantiated and a mutator (*e.g.*, set) or accessor (*e.g.*, get) method of the representative object is invoked. When a new user, node, node group, etc. is added, a new, empty representative object (*e.g.*, a user object, node object, node group object, etc.) is instantiated and its attributes are then populated with the new user's, node's, node group's, etc. attributes.

1    The SCM 12 preferably stores these objects in the data repository 26. The

2    data repository 26 preferably is a LDAP repository that is maintained in the

3    secondary storage device 141 of the CMS 14. Accordingly, the SCM 12 preferably

4    comprises an embodiment of the present invention, the persistent data manager 40,

5    as illustrated in Figure 2. The persistent data manager 40 is a software component

6    that stores objects 48 in and reads the stored objects 48 from the LDAP data

7    repository 26. Preferably, the LDAP data repository 26 is password-protected so

8    that only the persistent data manager 40 may write to the LDAP data repository 26.

9    The persistent data manager 40 acts as an interface to the LDAP data repository 26

10   for various consumers, such as object managers 44.

11   Referring to Figure 2, in an embodiment of the present invention, a plurality

12   of object managers 44 corresponding to each type of object (*e.g.*, user, node, node

13   group, etc.) interact with and direct the persistent data manager 40 to store the

14   objects 48 in and read the stored objects 48 from the LDAP data repository 26.

15   Object managers 44 preferably are software components that manage the objects 48,

16   providing access to the objects 48 to various consumers and instantiating the objects

17   when necessary.

18   Since objects 48 are defined by their attributes, the persistent data manager

19   40 stores objects 48 by storing persistent attributes of the objects 48 in the LDAP

20   data repository 26. Accordingly, when directed to store an object 48 in the LDAP

21   data repository 26, the persistent data manager 40 stores the persistent attributes in

22   the LDAP data repository 26. Likewise, when directed to read a stored object 48

23   from the LDAP data repository 26, the persistent data manager 40 reads the

24   persistent attributes from the LDAP data repository 26.

25   The persistent data manager 40 preferably stores the persistent attributes by

26   mapping the persistent attributes to certain defined LDAP attributes. These LDAP

27   attributes are preferably grouped within LDAP objects (not shown) that correspond

28   to the objects 48. These LDAP objects and their attributes are defined within the

29   LDAP data repository 26. In an embodiment of the present invention, the persistent

30   data manager 40 is responsible for defining, in the LDAP data repository 26, most

31   of the LDAP attributes and the LDAP objects that correspond to the objects 48.

32   Some of the LDAP attributes may be standard LDAP attributes that already exist.

33   These LDAP attributes and objects, referred to as persistent data schema, may be

1  defined and created using known LDAP methods for creating LDAP attributes and

2  objects.

3      The persistent data manager 40 may map the persistent attributes of the

4  objects 48 to the LDAP attributes utilizing a simple mapping methodology. For

5  example, the created LDAP attributes may be defined to have the same names as the

6  persistent attributes with a simple prefix (or suffix or other notation) indicating a

7  source (e.g., an organization and an application) from which the persistent attributes

8  originate. For example, a time stamp attribute of an object 48, named

9  "CreatedTime" may map to a created LDAP attribute named "hpmxCreatedTime."

10  The prefix "hpmx" may indicate, for example, that the source organization is

11  Hewlett-Packard Co.® and the source application is the SCM 12 (also known as

12  MX or MUXPlex). As noted above, the persistent data manager 40 may map some

13  of the persistent attributes to existing standard LDAP attributes (e.g.,

14  commonName, description, ipHostNumber, host and uidNumber).

15      The LDAP data repository 26 preferably includes a standard LDAP

16  directory structure. In an embodiment of the present invention, the persistent data

17  manager 40 sets up the LDAP data repository 26, creating a directory structure for

18  the source. For example, a file (e.g., a text file) may be sent to the persistent data

19  manager 40 describing and defining the source application (e.g., SCM 12), which

20  may be used for a main LDAP container, various container names, which may be

21  sub-containers of the source application container and are preferably identified by

22  object types, various LDAP objects, which preferably correspond to the objects 48

23  and are organized by the container names, and the LDAP attributes of these LDAP

24  objects, which preferably correspond to the persistent attributes. The persistent data

25  manager 40 may then parse the above information out of the file and set-up the

26  LDAP data repository 26, creating the directory structure, the LDAP objects and

27  LDAP attributes, using known, LDAP methods. A LDAP data repository 26 may

28  be set up in this manner for each application for which it is used. Figure 3

29  illustrates an exemplary directory structure or tree of a LDAP data repository 26 for

30  the SCM 12 set up at the source organization Hewlett-Packard, Inc.®.

31      In the example shown, the LDAP data repository 26 is accessible via the

32  Internet. Accordingly, the first level of the directory tree is o=hp.com, indicating a

33  web-accessible directory of the Hewlett-Packard, Inc.® organization. The LDAP

1    data repository 26 may be directly accessible from the first level of the directory

2    tree. Accordingly, in the example shown, the LDAP data repository 26 is indicated

3    by the second level named hpmxName="BMC Data Center." The LDAP syntax for

4    the second level path or distinguished name ("dn") is, therefore, dn = hpmxName =

5    BMC Data Center, o=hp.com.

6       In Figure 3, the SCM 12 is referred to by an application commonName

7    ("cn") "SysMgmt". Accordingly, a branch of the third level of the directory tree,

8    identifying the source application, is named cn=SysMgmt, indicating that the

9    objects 48 of the SCM 12 are stored, as described herein, within the SysMgmt

10    container of the LDAP data repository 26. In other words, the LDAP objects and

11    attributes to which the attributes of the objects 48 map are located within the

12    SysMgmt container. The SysMgmt container preferably is password protected so

13    that only the persistent data manager 40 may write to it. The LDAP syntax for this

14    third level branch path is the distinguished name cn=SysMgmt,hpmxName=BMC

15    Data Center, o=hp.com.

16       Preferably, the objects 48 are stored within different containers determined

17    by the type of object 48. In other words, the LDAP objects and attributes to which

18    the attributes of the objects 48 map are located within different containers

19    determined by the type of the object 48. For example, as described above, the SCM

20    12 may comprise node, user, node group, tool and role objects. Accordingly, the

21    directory tree shown in Figure 3 includes a fourth level comprising containers (also

22    known as groupOfNames). The containers shown include the following:

23    cn=NodeContainer (for node objects); cn=UserContainer (for user objects);

24    cn=NodeGroupContainer (for node group objects); cn=ToolContainer (for tool

25    objects); and, cn=RoleContainer (for role objects). The LDAP syntax for the path

26    of these containers is illustrated by the distinguished name

27    cn=NodeContainer,cn=SysMgmt,hpmxName=BMC Data Center, o=hp.com.

28       Referring to Figure 3, the fifth level of the directory tree shown includes

29    specific LDAP objects corresponding to existing objects 48 that have been stored in

30    the LDAP data repository 26 (*i.e.*, existing objects 48 that have had their attributes

31    mapped to LDAP attributes). Figure 3 illustrates two exemplary objects for each

32    container (*i.e.*, for each object type). For example, the container cn=NodeContainer

33    is shown including LDAP objects cn=hoffman and cn=windcave. These LDAP

34    objects correspond to node objects 48 named hoffman and windcave. The LDAP

1  syntax for the path of these LDAP objects is illustrated by the distinguished name

2  cn=hoffman,cn=NodeContainer,cn=SysMgmt,hpmxName=BMC Data

3  Center,o=hp.com.

4       Figures 4a and 4b illustrate the interaction of the persistent data manager 40,

5  object managers 44 (*e.g.*, NodeGroup Manager and Tool Manager), objects 48 (*e.g.*,

6  NodeGroup and Tool objects) and certain interfaces and other classes implemented

7  when one of the object managers 44 (*e.g.*, NodeGroup Manager) requests the

8  persistent data manager 40 to store or read one of the objects 48 (*e.g.*, NodeGroup

9  object) to or from the LDAP data repository 26. More specifically, Figures 4a and

10  4b illustrate an exemplary static structure diagram 60 of an persistent data manager

11  ("PDM") implementation class 62 (*e.g.*, MxPersistentDataManager) that is the

12  public interface for consumers (*e.g.*, object managers 44) of the persistent data

13  manager 40, a persistent object interface 64 (*e.g.*, MxPersistentObjectIfc) that

14  preferably is implemented by any object 48 that is to be stored in the LDAP data

15  repository 26, a LDAP utility class 66 (*e.g.*, LDAPConnection) that the persistent

16  data manager 40 uses to invoke standard LDAP methods for manipulating the

17  LDAP data repository 26, a series of reflection classes 68 that the persistent data

18  manager 40 uses to get or set the certain persistent attribute values from the objects

19  48, and object manager utility classes 70 and object implementation classes 72 that

20  are instantiated as object managers 44 and objects 48.

21       The LDAP utility class 66 preferably comprises standard LDAP methods for

22  adding, reading, modifying and deleting LDAP entries (*e.g.*, LDAP objects and

23  attributes) from a LDAP repository. The static structure diagram 60 also includes a

24  data repository interface 74 (*e.g.*, MxPersistentDataManagerIfc), as seen in Figure

25  4b, that may be implemented if the data repository 26 is not an LDAP repository,

26  but is some other type of repository (*e.g.*, an Oracle® database). If the data

27  repository 26 is, for example an Oracle database, the Oracle database preferably

28  implements the MxPersistentDataManagerIfc.

29       The exemplary static structure diagram 60 in Figures 4a and 4b include

30  subsets of the methods of the exemplary classes and interfaces illustrated. For

31  example, the PDM implementation class 62 may include additional methods beyond

32  those shown in Figure 4a. Likewise, the PDM implementation class 62 may include

33  less or different methods than the methods shown in Figure 4a. Moreover, the

1  names of the methods shown are not crucial to the implementation of the persistent

2  data manager 40 and different names may be used for the same methods without

3  deviating from the scope of the invention. In other words, the static structure

4  diagram 60 is exemplary in nature.

5  The PDM implementation class 62 preferably comprises private methods,

6  indicated by the "-" in front of the method names, and public methods, indicated by

7  the "+" in front of the method names. The persistent data manager 40 uses the

8  private methods internally. For example, the –connectToServer() method may be

9  invoked by the persistent data manager 40 to connect to the LDAP data repository

10  26. The –connectToServer() method, when invoked, may in turn invoke a LDAP

11  utility class 66 LDAP method +connect() to connect to the LDAP data repository

12  26. Likewise, a –disconnectFromServer() method may be invoked by the persistent

13  data manager 40 to disconnect from the LDAP data repository 26 in the same

14  manner. An –authenticateToServer() may be used to pass a LDAP data repository

15  26 password when the LDAP data repository 26 is password protected, as described

16  above.

17  The public methods of the PDM implementation class 62 are preferably

18  invoked by object managers 44 in order to store, delete, read, etc. the objects 48 in

19  the LDAP data repository 26. Associated with each of these public methods is

20  coding necessary to execute the processes described herein for performing these

21  tasks. As Figure 4a illustrates by the method invocation lines 75, the object

22  managers 44 (represented by the object manager utility classes 70) invoke a public

23  method of the PDM implementation class 62 when seeking to store an object 48 in

24  the LDAP data repository 26. The public method of the PDM implementation class

25  62 in turn uses the objects 48 (represented by the object implementation classes 72),

26  the series of reflection classes 68, and the LDAP utility class 66 to determine the

27  names of the objects' 48 attributes that are supposed to be stored in the LDAP data

28  repository 26 (*i.e.*, the persistent attributes described above), to reflect the persistent

29  attributes' values and to store the persistent attributes' values in the LDAP data

30  repository 26. The reading of objects 48 from the LDAP data repository 26 is

31  preferably implemented in a similar way, as described below.

32  As noted above, objects 48 that are to be stored in the LDAP data repository

33  26 preferably implement the persistent object interface 64. By implementing this

34  persistent object interface 64, the objects 48 will return data requested by the "get"

1   methods of the persistent object interface 64 when the persistent data manager 40

2   invokes these get methods. Accordingly, when an invoked public method of the

3   PDM implementation class 62 uses the objects 48, as described above and shown in

4   Figure 4a, the persistent data manager 40 preferably invokes the persistent object

5   interface 64 get methods shown in Figure 4b. The invoked get methods return data

6   identifying the persistent attributes, as well as data used to determine the LDAP data

7   repository 26 directory path or dn for the objects 48 being stored. In other words,

8   the persistent data manager 40 uses "reflection" on the objects 48 to get the objects

9   48 to "look in a mirror" and provide the requested data. Reflection is a known Java

10   technique to determine an object's attributes in this manner.

11       For example, the getPersistentAttrs() : String returns a lists of the persistent

12   attributes for the object 48 to be stored, the getApplicationName() : String returns a

13   string identifying the source application (*e.g.*, SysMgmt for the SCM 12)

14   corresponding to the third level of the directory tree shown in Figure 3, the

15   getCategoryName() : String returns a string identifying the container (*e.g.*, Node

16   Group Container) corresponding to the fourth level of the directory tree shown in

17   Figure 3 and getCommonName() : String returns a string identifying an attribute of

18   the object 48 that contains the name of the object 48. Preferably, the persistent data

19   manager 40 uses a reflection (represented by a reflection class 68) to get the value

20   of this name attribute of the object 48. The value of this name attribute of the object

21   48 (*i.e.*, the name of the object 48) corresponds to a LDAP object in the to the fifth

22   level of the directory tree shown in Figure 3.

23       The persistent attributes and dn are dynamically determined each time an

24   object 48 is stored in the LDAP data repository. By dynamically determining the

25   persistent attributes and dn, the persistent data repository can store any object 48

26   passed to it. The persistent data manager 40 may be coded with the persistent

27   attributes of each type of object 48. However, this would bloat the persistent data

28   manager 40 with excess code and require that the persistent data manager 40 be re-

29   coded any time a new object type was created or an existing object type modified

30   (*e.g.*, if additional attributes were added to the persistent attributes for a type of

31   objects 48). By dynamically determining the persistent attributes, these

32   disadvantages are avoided.

33       As noted above, the persistent data manager 40 uses reflection to retrieve the

34   values of the persistent attributes. Accordingly, invoked public methods, of the

PDM implementation class 62, instantiate the reflection classes 68 shown in order to reflect these values. Referring again to Figure 4a, the reflection::Field reflection class 68 represents a reflection used to get the data type of each of the persistent attributes. Accordingly, a getType() method is invoked for each certain persistent attribute, with the name of each certain persistent attribute passed with the method invocation, to retrieve the data type of the certain persistent attribute. The data type is needed to avoid exceptions for mismatched data types (*e.g.*, the persistent data manager 40 gets a String when a Long or Int is expected).

Referring to Figure 4a, the reflection::Method reflection class 68 represents a reflection used to get method names that are invoked to get or set the persistent attributes of the objects 48. It is preferably a matter of convention that each of the objects 48 includes a pair of methods that allow the persistent data manager 40 to get or set the persistent attributes' values by adding the words "get" or "set" to each persistent attributes' name. Accordingly, a getMethod() method is invoked for each certain persistent attribute of an object 48, with the name of each certain persistent attribute passed with the method invocation, to generate series of get or set method objects (depending on whether a store or read is intended) for all of the persistent attributes. Once the get or set method objects are returned to the persistent data manager 40, the get or set methods are invoked to get or set the persistent attributes' values from or to the object 48.

Referring to Figure 4a, the reflection::Array reflection class 68 represents a reflection used create an array for those persistent attributes that have multiple values (*e.g.*, that are arrays). Since an array is an object in Java, the reflection::Array reflection class 68 comprises a newInstance() method that is invoked to create a new instance of an array object. Invoking the appropriate get or set method may then fill in the instantiated array object. The filled-in array object is passed to the object 48 or the LDAP data repository, depending on whether a read or store of the object 48 is intended.

Figures 5a and 5b are flow-charts illustrating exemplary methods for storing and reading Java objects in a LDAP repository. As shown in Figure 5a, the method 80 for storing Java objects in a LDAP repository comprises starting a persistent data manager 81, requesting that an object 48 be stored 82, retrieving a list of the object's attributes 83, determining a repository path 84, connecting to the repository 85, retrieving the object's attribute values 86 and storing the object in the repository

87.  Starting a persistent data manager 81 preferably comprises an object manager 44 instantiating the PDM implementation class 62 or otherwise initiating the persistent data manager 81.  The PDM implementation class 62 may be instantiated, for example, by invoking a PDM implementation class 62 constructor (*e.g.*, MxPersistentDataManager()).

Requesting that an object be stored 82 preferably comprises an object manager 44 requesting that a persistent data manager 40 store an object 48.  The object manager 44 may accomplish this by invoking a PDM implementation class 62 method addEntry() (or "addObject") seen in Figure 4a and passing the object 48 with the method invocation.  When invoked, the addEntry() method preferably starts a process to store the object 48 in the LDAP data repository 26 and invokes the method steps described below.

Retrieving a list of the object's attributes 83 is preferably a step in the process started by the addEntry() method invocation; retrieving the list of the object's attributes 83 may be the first step.  The list of the object's attributes preferably comprises the names of only those attributes of the object 48 that are to be stored in the LDAP data repository (*i.e.*, the persistent attributes).  Therefore, retrieving the list of the object's attributes 83 comprises the persistent data manager 40 retrieving the persistent attributes from the object 48, preferably by invoking the persistent object interface 64 method to get the persistent attributes (*e.g.*, getPersistentAttrs()).  As noted above, an object 48 being stored in the LDAP data repository 26 preferably implements the persistent object interface 64 so that this method invocation will return the persistent attributes.

Determining a repository path 84 comprises determining the path, or dn, in the LDAP data repository 26 of the LDAP object that corresponds to the object 48 that is being stored.  Since the path, or dn, of the LDAP object is preferably determined from information in the object 48, determining a repository path 84 preferably comprises the persistent data manager 40 retrieving the information from the object 48.  Preferably, the persistent data manager 40 retrieves this information by invoking persistent object interface 64 methods to get the application name (*e.g.*, getApplicationName()), to get the container or category name (*e.g.*, getCategoryName()), to get the name of the object 48 (*e.g.*, getCommonName()).

As discussed above, these methods may actually retrieve the name of an attribute(s) of the object 48 that contains the object name, container name and/or

1 application name. In such an implementation, the persistent data manager 40 may

2 then retrieve the actual name(s) from the attribute(s) using reflection. The path, or

3 dn, in a LDAP data repository 26 at Hewlett-Packard Co. Inc. may then be

4 determined as dn = cn = CommonName (*e.g.*, hoffman), cn = CategoryName (*e.g.*,

5 NodeContainer), cn = ApplicationName (*e.g.*, SysMgmt), hpmxName = BMC Data

6 Center, o=hp.com. The execution of steps 83 and 84 are basically the object 48

7 entering a contract with the persistent data manager 40 to store the persistent

8 attributes at the determined dn in the LDAP data repository 26, and may be

9 performed in any order (*e.g.*, step 84 and then step 83).

10 Connecting to the repository 85 preferably comprises the repository data

11 manager 40 connecting to the LDAP data repository 26. The repository data

12 manager 40 may connect to the LDAP data repository 26 by invoking an internal

13 connect to server method (*e.g.*, connectToServer()), which in turn invokes a LDAP

14 utility class constructor method (*e.g.*, LDAPConnection()) instantiating the LDAP

15 utility class 66, which in turn invokes a LDAP method (*e.g.*, connect()) to connect

16 to the LDAP data repository 26. The connecting step 85 need not precede all of the

17 following steps, but the connecting step 85 should precede storing the object in the

18 repository 87.

19 Retrieving the object's attribute values 86 preferably comprises the

20 persistent object manager 40 getting the value of each persistent attribute of the

21 object 48 that is being stored. Preferably, the persistent object manager 40 retrieves

22 the object's attributes' values using the reflection technique, described above with

23 reference to Figures 4a and 4b. For example, the data type of each of the persistent

24 attributes is determined via reflection, a get method object for each of the certain

25 persistent attribute is created via reflection and the get method for each persistent

26 attribute in the object 48 is invoked, thereby retrieving the value of each persistent

27 attribute. If a persistent attribute is an array, an array object is instantiated and the

28 values are retrieved into the instantiated array object.

29 Once the persistent attributes of the object 48 being stored are retrieved, the

30 object 48 may be stored in the LDAP data repository 26. The persistent data

31 manager 40 may map the persistent attributes to LDAP attributes by appending the

32 simple pre-fix (or suffix or other notation) indicating the source onto the persistent

33 attributes' names, as described above. For example, using the prefix described

34 above, a createdBy persistent attribute maps to the LDAP attribute hpmxcreatedBy.

1  Accordingly, storing the object 48 in the repository 87 preferably comprises the

2  repository data manager 40 passing the LDAP object and the dn for the LDAP

3  object by mapping the persistent attributes to LDAP attributes, populating the

4  LDAP attributes with the persistent attributes' value and passing the LDAP

5  attributes and the dn to the LDAP data repository 26. The persistent data manager

6  40 may achieve this by directly invoking a LDAP API or by invoking a LDAP

7  uitlity class 66 method to add an entry to the LDAP data repository 26 (*e.g.*, add())

8  and passing the LDAP object and dn for the LDAP object with this method

9  invocation. The LDAP utility class 66 method in turn invokes the LDAP API to

10  store the mapped object 48 in the LDAP data repository 26.

11  As shown in Figure 5b, the method 90 for reading Java objects from a

12  LDAP repository comprises starting a persistent data manager 91, requesting that an

13  object 48 be read 92, retrieving a list of the object's attributes 93, determining a

14  repository path 94, connecting to the repository 95, retrieving the object's attribute

15  values from the repository 96 and setting the object's attribute values 97. Starting a

16  persistent data manager 91 may be accomplished as described above with reference

17  to Figure 5a.

18  Requesting that an object 48 be read 92 preferably comprises an object

19  manager 44 requesting that a persistent data manager 40 retrieve or load the object

20  48 from a LDAP data repository 26. The object manager 44 may accomplish this

21  by invoking a PDM implementation class 62 method readEntry() (or "readObject")

22  seen in Figure 4a and passing an empty object 48 with the method invocation.

23  When invoked, the readEntry() method preferably starts a process to read the object

24  48 from the LDAP data repository 26 and invokes the method steps described

25  below.

26  Retrieving a list of the object's attributes 93 is preferably a step in the

27  process started by the readEntry() method invocation; retrieving the list of the

28  object's attributes 93 may be the first step in this process. The list of the object's

29  attributes preferably comprises the names of only those attributes of the object 48

30  that are stored in the LDAP data repository 26 (*i.e.*, the persistent attributes).

31  Therefore, retrieving the list of the object's attributes 93 comprises the persistent

32  data manager 40 retrieving the persistent attributes from the empty object 48,

33  preferably by invoking the persistent object interface 64 method to get the persistent

34  attributes (*e.g.*, getPersustentAttrs()). As noted above, an object 48 stored in the

1    LDAP data repository 26, in this case the empty object 48, preferably implements

2    the persistent object interface 64 so that this method invocation will return the

3    persistent attributes.

4        Determining a repository path 94 comprises determining the path, or dn, in

5    the LDAP data repository 26 of the LDAP object that corresponds to the empty

6    object 48. Since the path, or dn, of the LDAP object is preferably determined from

7    information in the empty object 48, determining a repository path 94 preferably

8    comprises the persistent data manager 40 retrieving the information from the empty

9    object 48. Preferably, the persistent data manager 40 retrieves this information by

10   invoking persistent object interface 64 methods to get the application name (*e.g.*,

11   getApplicationName()), to get the container or category name (*e.g.*,

12   getCategoryName()), to get the name of the object 48 (*e.g.*, getCommonName()).

13       As discussed above, these methods may actually retrieve the name of an

14   attribute(s) of the empty object 48 that contains the object name, container name

15   and/or application name. In such an implementation, the persistent data manager 40

16   may then retrieve the actual name(s) from the attribute(s) using reflection. The

17   path, or dn, in a LDAP data repository 26 at Hewlett-Packard Co. may then be

18   determined as dn = cn = CommonName (*e.g.*, hoffman), cn = CategoryName (*e.g.*,

19   NodeContainer), cn = ApplicationName (*e.g.*, SysMgmt), hpmxName = BMC Data

20   Center, o=hp.com. The execution of steps 93 and 94 are basically the object 48

21   entering a contract with the persistent data manager 40 to read the persistent

22   attributes from the determined dn in the LDAP data repository 26, and may be

23   performed in any order (*e.g.*, step 94 and then step 93).

24       Connecting to the repository 95 may be accomplished as described above

25   with reference to Figure 5a. Referring again to Figure 5b, retrieving the object's

26   attribute values from the repository 96 preferably comprises the persistent data

27   manager 40 reading the empty object's 48 persistent attributes' values from the

28   LDAP data repository 26. This may comprise the persistent data manager 40

29   mapping the persistent attributes to LDAP attributes, as described above, passing

30   the dn for the LDAP object, determined as described above in steps 93 and 94, to

31   the LDAP data repository 26 and populating the mapped persistent attributes with

32   the values of the corresponding LDAP attributes stored in the LDAP data repository

33   26. The persistent data manager 40 may achieve this by directly invoking a LDAP

34   API or by invoking a LDAP uitlity class 66 method to read an entry from the LDAP

1    data repository 26 (*e.g.*, read()) and passing the dn for the LDAP object with this

2    method invocation. The LDAP utility class 66 method in turn invokes the LDAP

3    API to read the LDAP attributes' values located by the dn from the LDAP data

4    repository 26.

5         Once the values of the LDAP attributes corresponding to the persistent

6    attributes of the empty object 48 have been read from the LDAP data repository 26,

7    the empty object 48 may be populated with these values. Setting the object's

8    attribute values 97 may comprise the persistent data manager 40 populating the

9    persistent attributes with the LDAP attributes' values. As with the method

10   described above with regard to Figure 5a, Java reflection, or a similar technique, is

11   preferably used to get the data types and method names of the persistent attributes.

12   For setting the object's attribute values 97, the method names are "set" methods,

13   since the persistent attributes are being set with the LDAP attributes' values. As

14   above, if any of the persistent attributes are arrays, a reflection is preferably used to

15   create an instance of an array object. Once the data types and method names are

16   retrieved, the persistent data manager 40 preferably invokes the set methods to set

17   each of the persistent attributes of the empty object 48 and populate the empty

18   object 48.

19        Figures 6a and 6b are exemplary sequence diagrams for storing a node group

20   object in a LDAP data repository and reading a node group object from a LDAP

21   data repository. These sequence diagrams illustrate exemplary methods for storing

22   and reading objects to and from an LDAP data repository; methods similar to the

23   method illustrated by these exemplary sequence diagrams may be used to store and

24   read different types of objects to and from an LDAP data repository. Each sequence

25   diagram comprises: boxes 102, each representing a class; vertical time-lines 104,

26   each representing continuous running of a class; horizontal method call-lines 106,

27   each representing a first class invoking a target class method; and notation boxes

28   108 including comments about the sequence diagrams.

29        Figure 6a is an exemplary sequence diagram 100 for storing a node group

30   object 48 in a LDAP data repository 26. When the node group object 48 is stored in

31   a LDAP data repository 26 a LDAP node group object is created. The first method

32   call-line 106, issuing from a vertical time-line 104 descending from a

33   mxNodeGroupMgr box 102 to a mxPersistentDataManager box 102, represents a

34   node group manager 44 starting the persistent data manager 40 by invoking a PDM

1 implementation class constructor (*e.g.*, MxPersistentDataManager()). After the

2 PDM implementation class is instantiated, the node group manager 44 requests that

3 the persistent data manager 40 store a node group object 48 in the LDAP data

4 repository 26, as shown by an addEntry() method call-line 106. The node group

5 manager 44 preferably passes the node group object 48 to the persistent data

6 manager 40 with the addEntry() method invocation.

7 The invoked addEntry() method in turn invokes a process to store the node

8 group object 48 in the LDAP data repository 26. As part of the process, the

9 attributes of the node group object 48 that are to be stored in the LDAP data

10 repository (*e.g.*, the persistent attributes) are determined. Likewise, the path or dn

11 of the LDAP object to which the node group object 48 will be mapped is

12 determined. As noted above, the persistent attributes and dn are dynamically

13 determined for each object 48. The getApplicationName(), getCategoryName(),

14 getCommonName() and getPersistentAttrs() method call-lines 106 represent

15 methods of the node group object 48 invoked by the persistent data manager 40 to

16 dynamically determine the dn and persistent attributes. In response to these method

17 invocations, the node group object 48 returns the node group object 48 attributes

18 with the application name (*e.g.*, SysMgmt), the category or container name (*e.g.*,

19 node group container) and the common name of the node group object 48 (*e.g.*,

20 Oracle Server Group), in order to build the dn, as well as the persistent attributes.

21 As described above, these methods may be persistent object interface 64 methods

22 implemented by the node group object 48. The order of execution of these methods

23 may be changed without deviating from the scope of the invention.

24 Referring again to Figure 6a, a LDAPConnection() method call-line 106

25 indicates that the persistent data manager 40 instantiates the LDAP utility class 66.

26 A getType() method call-line 106 indicates that the persistent data manager 40

27 invokes a Reflection::Field class method to get the data type of a node group object

28 48 persistent attribute. The data type for each of the persistent attributes is retrieved

29 using reflection. Accordingly, the persistent data manager 40 repeats the getType()

30 method invocation until the data type for each of the persistent attributes is

31 retrieved. The dashed-line pointing from Reflection::Field class time-line 104 to the

32 PDM implementation class 62 time-line 104 indicates that the Reflection::Field

33 class returns control to the persistent data manager 40.

1    A getMethod() method call-line 106 indicates that the persistent data

2    manager 40 invokes a Reflection::Method class method to get a method name of the

3    node group object 48 get method to retrieve one of the persistent attributes. Method

4    names for get methods to retrieve each of the persistent attributes are retrieved using

5    reflection. Accordingly, the persistent data manager 40 repeats the getMethod()

6    method invocation until the method name for each of the persistent attributes is

7    retrieved.

8        A getMemberObjects() method call-line 106 indicates that the persistent

9    data manager 40 invokes the get methods, retrieved above, to get the values of all of

10   the persistent attributes of the node group object 48. Once all the persistent

11   attributes of the node group object 48 are retrieved, the persistent data manager 40

12   can store the node group object 48 in the LDAP data repository 26 by mapping the

13   persistent attributes to corresponding LDAP attributes of a LDAP object. Referring

14   to Figure 6a, an add() method call-line 106 indicates that the persistent data

15   manager 40 invokes a LDAP utility class 66 method to add the node group object

16   48 to the LDAP data repository 26. The persistent data manager 40 passes the

17   mapped persistent attributes' values, and the dn determined above, to the LDAP

18   utility class 66 with the add() method invocation. The LDAP utility class 66

19   preferably executes the LDAP API(s) necessary to store the mapped persistent

20   attributes' values in the LDAP data repository 26.

21       As shown by the dashed-line labeled add() pointing from the LDAP utility

22   class time-line 104 to the persistent data manager 40 time-line 104, and the

23   addEntry() dashed-line pointing from the persistent data manager time-line 104 to

24   the node group manager time-line 104, the LDAP utility class 66 returns control to

25   the persistent data manager 40, which in turn returns control to the node group

26   manager 44, when the process of storing the node group object 48 in the LDAP data

27   repository 26 is complete.

28       Figure 6b is an exemplary sequence diagram 101 for reading a stored node

29   group object from a LDAP data repository 26. The persistent attribute values of an

30   empty node group object 48 would have been previously stored in corresponding

31   LDAP attributes in the LDAP data repository 26. Much of the method invocations

32   are the same in the sequence diagram 101 as in the sequence diagram 100 for

33   storing a node group object 48. In Figure 6b, the node group manager 44 invokes a

34   PDM implementation class 62 method to read the node group object 48 entry in the

1     LDAP data repository 26, as indicated by the readEntry() method call-line 106. The

2     invoked readEntry() method in turn invokes a process retrieve the node group object

3     48 persistent attribute values from the LDAP data repository 26.

4         As a part of that process, the persistent data manager 40 invokes the node

5     group 48 methods to determine the persistent attributes and the dn for the persistent

6     attribute values in the LDAP data repository 26. The getApplicationName(),

7     getCategoryName(), getCommonName() and getPersistentAttrs() method call-lines

8     106 represent these invoked methods. After instantiating LDAP utility class 66, the

9     persistent data manager 40 invokes a LDAP utility class 66 method to read the

10     persistent attribute values from the LDAP data repository 26. This is indicated by

11     read() method call-line 106. The persistent data manager 40 preferably passes the

12     dn for the persistent attribute values (*i.e.*, the dn for the LDAP attributes mapped to

13     by the persistent attributes) to the LDAP utility class 66 with the read() method

14     invocation. The LDAP utility class 66 preferably executes the LDAP API(s)

15     necessary to read the persistent attribute values from the location identified by the

16     dn.

17         Once the persistent attribute values are retrieved, the persistent data manager

18     40 retrieves the data type and method names for persistent attributes, as described

19     above and as indicated by the getType() and getMethod() method call-lines 106

20     shown in Figure 6b. In the process illustrated by the sequence diagram 101,

21     however, the method names are set methods, since the persistent attributes of the

22     node group object are being populated or set with the persistent attribute values

23     retrieved from the LDAP data repository 26.

24         Once the method names for the set methods of the persistent attributes have

25     been retrieved using reflection, the persistent data manager 40 sets all of the

26     persistent attributes using these set methods and the persistent attribute values

27     retrieved from the LDAP data repository 26. This is indicated by the

28     setMemberObjects() method call line 106 from the persistent data manager time-

29     line 104 to the node group time line 104. Once the persistent attributes of the node

30     group object have been set, the persistent data manager preferably returns control to

31     the node group manager 44.

32         The persistent data manager 40 can perform processes similar to those

33     shown in Figures 6a and 6b for adding and reading different types of Java objects

34     48 to and from the LDAP data repository 26. By dynamically determining the

1  persistent attributes and dn of the objects 48 and using reflection to determine the

2  values of the persistent attributes, the persistent data manager 40 maintains the

3  flexibility to handle numerous types of objects 48.

4  Some of the objects and classes discussed herein are named with a prefix

5  "mx". The mx prefix is indicative of the application utilizing the objects and

6  classes (*e.g.*, the SCM 12) and is merely exemplary. Indeed, the names of classes,

7  objects and methods discussed herein are exemplary, are not intended to be limiting,

8  and are merely used for ease of discussion.

9  While the invention has been described with reference to the exemplary

10 embodiments thereof, those skilled in the art will be able to make various

11 modifications to the described embodiments of the invention without departing from

12 the true spirit and scope of the invention. The terms and descriptions used herein

13 are set forth by way of illustration only and are not meant as limitations. Those

14 skilled in the art will recognize that these and other variations are possible within

15 the spirit and scope of the invention as defined in the following claims and their

16 equivalents.